

UNITED STATES PATENT APPLICATION

**PROCESSOR BASED SYSTEM AND METHOD FOR VIRUS DETECTION**

**INVENTORS:**

Kaustubh Das

Hani Elgebaly

Schwegman, Lundberg, Woessner, & Kluth, P.A.  
1600 TCF Tower  
121 South Eighth Street  
Minneapolis, Minnesota 55405  
SLWK: 884.934US1

# PROCESSOR BASED SYSTEM AND METHOD FOR VIRUS DETECTION

## LIMITED COPYRIGHT WAIVER

5 A portion of the disclosure of this patent document contains material to which the claim of copyright protection is made. The copyright owner has no objection to the facsimile reproduction by any person of the patent document or the patent disclosure, as it appears in the U.S. Patent and Trademark Office file or records, but reserves all other rights whatsoever.

## 10 FIELD

This invention relates generally to computer viruses and more particularly to the computer virus detection systems.

## BACKGROUND

15 In general, computer viruses (hereinafter referred to as viruses) are programs designed to replicate themselves by attaching virus programs to non-virus software. For example, a virus might attach a copy of itself to a spreadsheet program, word processing document, Internet browser, computer game, etc. After a program has been “infected” with a virus, each time the infected program runs the virus also  
20 runs, further replicating the virus. Because the presence of computer viruses often goes undetected, viruses can cause unexpected and harmful results. For example, viruses have been known to delete files, alter system settings, and consume system resources.

Traditionally, there have been two main virus types including executable and  
25 boot sector viruses. Executable viruses attach themselves to executable programs, so the virus programs run while the executable programs are running. One characteristic of executable viruses is that they will not execute until the “host” program is executed. Boot sector viruses attach themselves to floppy or hard disk boot sectors. Boot sectors store operating system programs for loading parts of an  
30 operating system into a computer’s memory during boot-up. When viruses are

stored in the boot sector, they are guaranteed to execute because boot sector programs are always executed during operating system boot-up. Once the boot sector virus is loaded into memory, it typically can infect the boot sector of any floppy disk inserted into the computer.

5           Virus detection software has been developed to detect and eliminate these and other computer virus types. Virus detection programs typically scan computer files for specific bit patterns associated with known viruses. These bit patterns are often referred to as virus signatures. Scanning files for virus signatures can be a slow and resource draining process. Various techniques have been developed to  
10       limit the scope of signature searches. One such technique is scalpel scanning, which limits signature searching to the parts of file that are likely to contain virus entry points.

              However, virus writers have thwarted many signature-scanning techniques by creating randomly encrypted and polymorphic viruses. Randomly encrypted  
15       viruses are difficult to detect because each new copy of the virus is randomly encrypted, so new virus copies may not exhibit traceable signatures until they are decrypted. Randomly encrypted viruses remain encrypted until just before execution, when they perform self-decryption, which may reveal known signatures. Polymorphic viruses are also difficult to detect because they change their encryption  
20       logic with each new infection. That is, the virus produces different encrypting and decrypting code for each new virus that is inserted into non-virus software. Because the encryption/decryption code is constantly changing, copies of the virus may not include traceable signatures, even when the virus is not encrypted.

              In response to random encryption and polymorphic viruses, some virus  
25       detection systems emulate executable programs in secure portions of memory. Because encrypted viruses decrypt themselves before executing, emulating potentially infected programs can produce viruses in a decrypted state. Matching decrypted viruses with known virus signatures is typically more effective than doing the same with encrypted viruses. During emulation, the emulator periodically scans  
30       the secure memory portion for known virus signatures. If the emulator finds known

virus signatures, the corresponding non-virus programs are processed and viruses are removed.

One disadvantage of using emulators to search for virus signatures is that emulators consume a relatively large amount of system resources. Another  
5 disadvantage is that emulators can miss known viruses when the viruses execute before being processed by the emulator. Yet another disadvantage is that some viruses are “aware” of emulators and thus do not decrypt during emulation. Another disadvantage is that emulators often do not support an entire processor instruction set. Thus, an emulator may not detect viruses that include instructions which the  
10 emulator does not support.

Another disadvantage of emulator based virus detection systems is that emulators typically do not know how long to emulate programs before associated viruses will decrypt themselves. Because emulation times are unknown, the only way to ensure that emulation times are not too short for decrypting viruses is to  
15 emulate programs forever, which is typically impossible.

#### BRIEF DESCRIPTION OF THE FIGURES

The invention may best be understood by referring to the following description and accompanying drawings that are used to illustrate embodiments of  
20 the invention. In the drawings:

**Figure 1** illustrates an exemplary computer system used in conjunction with certain embodiments of the invention;

**Figure 2** is a block diagram illustrating an architecture for a processor-based virus detection system, according to embodiments of the invention;

25 **Figure 3** is a block diagram illustrating an alternative architecture for the processor based virus detection system of Figure 2, according to exemplary embodiments of the invention;

**Figure 4** is a block diagram illustrating an alternative architecture for the processor based virus detection system of Figure 2, according to exemplary  
30 embodiments of the invention;

**Figure 5** is a flow diagram illustrating operations of a virus detection unit, according to exemplary embodiments of the invention;

**Figure 6** is a flow diagram illustrating exemplary operations for fetching and executing an instruction, according to exemplary embodiments of the invention;

5       **Figure 7** is a flow diagram illustrating operations of a virus detection unit, according to exemplary embodiments of the invention;

**Figure 8** is a more detailed block diagram of the virus information unit of Figure 3, according to exemplary embodiments of the invention;

10       **Figure 9** is a more detailed description of operations for processing an instruction to determine whether the instruction is associated with a virus, as described in Figure 7, according to exemplary embodiments of the invention; and

**Figure 10** is a block diagram of a state machine, according to exemplary embodiments of the invention.

15       **Figure 11** is a flow diagram illustrating operations for using a state machine in determining whether an instruction is associated with a virus, according to exemplary embodiments of the invention.

**Figure 12** is a flow diagram illustrating operations for receiving a interrupt vector identifying a virus processing interrupt handler, according to exemplary embodiments of the invention.

20       **Figure 13** is a flow diagram illustrating operations for transmitting a interrupt vector identifying a virus processing interrupt handler, according to exemplary embodiments of the invention.

25       **Figure 14** is a flow diagram illustrating operations for transmitting virus information to a virus detection unit, according to exemplary embodiments of the invention.

**Figure 15** is a flow diagram illustrating operations for transmitting virus information to a virus detection unit, according to exemplary embodiments of the invention.

30       **Figure 16** is a data flow diagram illustrating the general behavior of a polymorphic virus.

## DESCRIPTION OF THE EMBODIMENTS

In the following description, numerous specific details are set forth. However, it is understood that embodiments of the invention may be practiced  
5 without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail in order not to obscure the understanding of this description.

Herein, block diagrams illustrate exemplary embodiments of the invention. Also herein, flow diagrams illustrate operations of the exemplary embodiments of  
10 the invention. The operations of the flow diagrams will be described with reference to the exemplary embodiments shown in the block diagrams. However, it should be understood that the operations of the flow diagrams could be performed by embodiments of the invention other than those discussed with reference to the block diagrams, and embodiments discussed with references to the block diagrams could  
15 perform operations different than those discussed with reference to the flow diagrams.

### Hardware and Operating Environment

This section provides an overview of the exemplary hardware and the  
20 operating environment in which embodiments of the invention can be practiced.

**Figure 1** illustrates an exemplary computer system used in conjunction with certain embodiments of the invention. As illustrated in Figure 1, computer system 100 comprises processor(s) 102, which includes a virus detection unit. Computer system 100 also includes a memory 132, processor bus 110, memory controller hub  
25 158, and input/output controller hub (ICH) 140. As shown in Figure 1, the processor(s) 102, memory controller hub 158, and ICH 140 are connected together. The processor(s) 102 may comprise any suitable processor architecture. The computer system 100 may comprise one, two, three, or more processors, any of which may execute a set of instructions in accordance with embodiments of the  
30 present invention.

The memory controller hub 158 provides an interface to the memory 132, which stores data and/or instructions, and may comprise any suitable memory, such as a dynamic random access memory (DRAM), for example. The computer system 100 also includes IDE drive(s) 142 and/or other suitable storage devices. A  
5 graphics controller 134 controls the display of information on a display device 137, according to embodiments of the invention.

The input/output controller hub (ICH) 140 provides an interface to I/O devices or peripheral components for the computer system 100. The ICH 140 may comprise any suitable interface controller to provide for any suitable communication  
10 link to the memory controller hub 158 and/or to any suitable device or component in communication with the ICH 140. For one embodiment of the invention, the ICH 140 provides suitable arbitration and buffering for each interface.

For one embodiment of the invention, the ICH 140 provides an interface to one or more suitable integrated drive electronics (IDE) drives 142, such as a hard  
15 disk drive (HDD) or compact disc read only memory (CD ROM) drive, or to suitable universal serial bus (USB) devices through one or more USB ports 144. For one embodiment, the ICH 140 also provides an interface to a keyboard 151, a mouse 152, a CD-ROM drive 155, one or more suitable devices through one or more parallel ports 153 (e.g., a printer), and one or more suitable devices through  
20 one or more serial ports 154. For one embodiment of the invention, the ICH 140 also provides a network interface 156 through which the computer system 100 can communicate with other computers and/or devices.

In one embodiment, the computer system 100 includes a machine-readable medium that stores a set of instructions (e.g., software) embodying any one, or all,  
25 of the methodologies described herein. Furthermore, software can reside, completely or at least partially, within memory 132 and/or within the processor(s) 102.

### System Level Overview

This section provides a system level overview of exemplary embodiments of the invention. Figures 2-4 show architectures for functional elements contained within a processor. Operations for the functional elements are described in the next section.

**Figure 2** is a block diagram illustrating an architecture for a processor-based virus detection system, according to embodiments of the invention. Figure 2 shows a processor 200 and its functional elements. The processor 200 includes a bus interface unit 204 coupled to a level-2 cache 202 and a system bus 222. The bus interface unit 204 is also coupled to a level-one instruction cache 206 and a level-one data cache 208. The level-one instruction cache 206 is coupled to a virus detection unit 212, which is coupled to a fetch and decode unit 214. The fetch and decode unit 214 is coupled to an instruction pool 220, which is coupled to a dispatch and execution unit 216 and a retirement unit 218. The retirement unit 218 is coupled to the level-one data cache 208 and a register pool 210. The processor 200 can also include functional elements and connections not shown in Figure 2. For example, the processor 200 can include additional arithmetic-logic units for performing various calculations, such as address calculations and other arithmetic operations. As another example, the processor 200 can include bus connectivity between the register pool 210 and the dispatch and execution unit 216.

**Figure 3** is a block diagram illustrating an alternative architecture for a processor based virus detection system, according to exemplary embodiments of the invention. Figure 3 is similar to Figure 2 except the virus detection unit 212 is shown in more detail. In the embodiment depicted in Figure 3, the virus detection unit 212 includes three constituent units including a virus detection engine 302, and authentication unit 306, and a virus information unit 304. Although not shown, the constituent units (302, 306, and 304) of the virus detection unit 212 are capable of various connectivities. For example, the constituent units can be fully connected or connected through any other suitable connection architecture. Moreover the

constituent units can communication according to any suitable communication method or protocol.

In one embodiment, the virus information unit 304 stores virus information including virus signatures and/or virus profiles including virus state information.

5 The virus detection engine 302 uses the virus information to determine whether instructions are associated with a virus. The authentication unit 306 authenticates the source of virus information and virus processing interrupt handlers (e.g., software programs residing in the memory 132 that transmit virus information to the virus detection unit 212). These units and operations performed by these units will  
10 be described in greater detail below.

According to embodiments of the invention, the virus detection engine 302, authentication unit 306, and virus information unit 304 can be various integrated circuits, memories, and/or machine-readable media for performing operations according to embodiments of the invention. Machine-readable media includes any  
15 mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory (ROM), random access memory (RAM), magnetic disk storage media, optical storage media, flash memory devices, electrical, optical, acoustical or other forms of propagated signals (e.g., carrier  
20 waves, infrared signals, digital signals, etc.), etc. According to embodiments of the invention, the virus detection engine 302, authentication unit 306, and virus information unit 304 can be other types of logic (e.g., digital logic) for executing the operations described herein.

**Figure 4** is a block diagram illustrating an alternative architecture for the  
25 processor based virus detection system of Figure 2, according to exemplary embodiments of the invention. Figure 4 is similar to Figures 2 and 3 except that the virus detection unit 212 is shown including a virus detection engine 402 and an authentication unit 404. As shown in Figures 2-4, the virus detection unit 212 can include different constituent units, according to alternative embodiments of the  
30 invention. It should be understood that the constituent units of the virus detection

unit 212 can be further subdivided or integrated, according to alternative embodiments of the invention. Moreover, it should be understood that the virus detection unit 212 can be coupled to different functional units of the processor 200. For example, the virus detection unit 212 can be coupled to the fetch and decode unit 214 and the instruction pool 220. Alternatively, the virus detection unit can be coupled to the instruction pool to the dispatch and execution unit 216.

### Methods of the Invention

This section describes methods performed by embodiments of the invention.

10 In certain embodiments, the methods are performed by machine-readable media (e.g., RAM), while in other embodiments, the methods are performed by hardware or other logic (e.g., digital logic). The discussion of Figures 2-4 above described architectures for the processor and its functional units, whereas the following discussion of Figure 5 will describe the operations of the processor's functional

15 units. In particular, Figure 5 describes the data flow and interaction between the processor's functional units.

**Figure 5** is a flow diagram illustrating operations of a virus detection unit, according to exemplary embodiments of the invention. The operations of the flow diagram 500 will be described with reference to the exemplary virus detection unit of Figure 2. The flow diagram 500 commences at block 502, where an instruction is received in a first functional unit of a processor pipeline. For example, referring to Figure 2, the virus detection unit 212 receives an instruction from the level-one instruction cache 206. According to alternative embodiments, the first functional units can be the fetch and decode unit 214 or instruction pool 220. In alternative

20

25 embodiments, the virus detection unit 212 receives instructions from still other functional units. The process continues at block 504.

At block 504, it is determined whether the instruction is associated with a virus. For example, the virus detection unit 212 determines whether the instruction is associated with a virus. The process continues at block 506.

As shown in block 506, after determining the instruction is not associated with a virus, the instruction is transmitted to a second functional unit of the processor pipeline for further processing. For example, after determining the instruction is not associated with a virus, the virus detection unit 212 transmits the instruction to the fetch and decode unit 214 for further processing. According to alternative embodiments, the second functional unit could be the instruction pool 220 or the dispatch and execution unit 216. From block 506, the process ends.

### Exemplary Implementation

This section describes exemplary embodiments of the invention in greater detail. In the following discussion, Figures 6-16 will be described. Figure 6 describes the general operations performed by the functional units of the processor 200 for fetching and executing an instruction, while Figure 7 describes operations of the virus detection unit 212.

**Figure 6** is a flow diagram illustrating exemplary operations for fetching and executing an instruction, according to exemplary embodiments of the invention. Figure 6 will be described with reference to the exemplary processor shown in Figure 2. The flow diagram 600 commences at block 602, where an instruction is requested from the level-one cache. For example, referring to Figure 2, the fetch and decode unit 214 requests an instruction from the level-one instruction cache 206. The process continues at block 604.

At block 604, an instruction is transmitted to the virus detection unit. For example, the level-one instruction cache transmits the requested instruction to the virus detection unit 212. The process continues at block 606.

As shown in block 606, the instruction is processed to determine whether it is associated with a virus. For example, the virus detection unit 212 determines whether the instruction is associated with a virus. Operations for determining whether an instruction is associated with a virus are described in greater detail below. The process continues at block 607.

At block 607, it is determined whether the instruction is associated with a virus. For example, the virus detection unit 212 determines whether the instruction is associated with a virus. If the instruction is associated with a virus, the process ends. Otherwise, the process continues at block 608.

5           At block 608, the instruction is transmitted to the fetch and decode unit. For example, the virus detection unit 212 transmits the instruction to the fetch and decode unit 214. From block 608, the process continues at block 610.

          As shown in block 610, the instruction is transmitted to the instruction pool. For example, the fetch and decode unit 214 transmits the instruction to the  
10   instruction pool 220. The process continues at block 612.

          At block 612, an instruction is selected and executed. For example, the dispatch and execution unit 216 selects and executes the instruction from the instruction pool 220. The process continues at block 614.

          At block 614, the instruction is transmitted back to the instruction pool. For  
15   example, the dispatch and execution unit 216 transmits the executed instruction back to the instruction pool 220. The process continues at block 616.

          As shown in block 616, the executed instruction is transmitted to the retirement unit. For example, the instruction pool 220 transmits the executed instruction to the retirement unit 218. The process continues at block 618.

20           At block 618, the executed instruction is retired. For example, the retirement unit 218 retires the executed instruction. In one embodiment, as part of instruction retirement, the retirement unit 218 updates registers in the register pool 210 and data in the level-one data cache 208.

          While Figure 6 illustrates the operations of the processor 200 as occurring  
25   sequentially for one instruction, it should be understood that the functional units of the processor 200 are capable of executing in parallel for multiple instructions. Therefore, the functional units of the processor 200 form a processor pipeline. For example, the fetch and decode unit 214 fetches an instruction, while the dispatch and execution unit 216 contemporaneously executes another instruction. Similarly,  
30   the virus detection unit 212 is capable of determining whether an instruction is

associated with a virus at the same time the retirement unit 218 is retiring a different instruction. Moreover, in one embodiment, the processor 200 is capable of executing instructions out-of-order and reordering the instructions before retirement.

5 While the discussion of Figure 6 above described the operations of many of the processor's functional units, the following discussion of Figure 7 will describe the general operations of the processor's virus detection unit. Certain of the operations set forth in Figure 7 will also be described in more detail below, with reference to Figures 8-15.

**Figure 7** is a flow diagram illustrating operations of a virus detection unit, according to exemplary embodiments of the invention. The operations of the flow diagram 700 will be described with reference to the exemplary virus detection units of Figures 2-4. The flow diagram 700 commences at block 702, where an instruction is received from the cache. For example, referring to Figure 2, the virus detection unit 212 receives an instruction from the level-one instruction cache 206.  
15 The process continues at block 704.

At block 704, the instruction is processed to determine whether it is associated with a virus. For example, the virus detection unit 212 determines whether the instruction is associated with a virus. In one embodiment, the virus detection engine 302 compares the instruction to virus signatures stored in the virus information unit 304. In an alternative embodiment, the virus detection engine 302 runs the instruction through a state machine, which uses state information stored in the virus information unit 304. The virus detection unit's operations for processing instructions are described in greater detail below (see Figures 8-11). The process continues at block 706.  
20

As shown in block 706, it is determined whether the instruction is associated with a virus. For example, the virus detection unit 212 determines whether the instruction is associated with a virus based on results of the processing performed at block 704. If the instruction is associated with a virus the process continues at block 710. Otherwise, the process continues at block 708.  
25

At block 708, the instruction is transmitted to the fetch and decode unit. For example, the virus detection unit 212 transmits the instruction to the fetch and decode unit 214. From block 708, the process continues at block 702.

As shown in block 710, virus processing and removal is performed. For example, the virus detection unit 212 causes the processor 200 to flush the processor pipeline and drop the instruction associated with a virus. In one embodiment, the virus detection unit 212 generates an interrupt, calling an interrupt handler (interrupt handlers are described in more detail below) to remove the virus instructions from the processor pipeline and any associated memory systems (e.g., cache, main memory, and secondary storage devices). From block 710, the process ends.

Figures 8-11 illustrate concepts and operations for determining whether instructions are associated with a virus, as described above in block 704 of Figure 7. In particular, Figures 8 and 9 describe signature matching, while Figures 10 and 11 describe operations of a state machine.

**Figure 8** is a block diagram illustrating a virus information unit, according to exemplary embodiments of the invention. As shown in Figure 8, the virus information unit 304 includes virus signatures. Virus signatures are bit patterns that appear in known viruses. For example, the virus signatures shown in Figure 8 represent hexadecimal byte strings of known viruses. In one embodiment, the virus information unit 304 includes a content addressable memory. Alternatively, the virus information unit 304 can be any other suitable fast memory. In one embodiment, the virus detection engine 302 compares instructions received from the level-one instruction cache 206 with the virus signatures stored in the virus information unit 302, as described below with reference to Figure 9.

**Figure 9** is a more detailed description of operations for processing an instruction to determine whether the instruction is associated with a virus, as described in Figure 7, according to exemplary embodiments of the invention. The flow diagram 900 will be described with reference to the exemplary processor of Figures 2-4. The flow diagram 900 commences at block 902, where one or more instructions are combined to form an instruction string. For example, the virus

detection engine 302 forms an instruction string including one or more instructions. The process continues at block 904.

At block 904, the instruction string is compared to the contents of the instruction information unit. For example, the virus detection engine 302 compares the instruction string to the contents of the virus information unit 304. The process continues at block 906.

At block 906, it is determined whether the instruction string matches virus signatures stored in the virus information unit. For example, the virus detection engine 302 determines whether the instruction string matched a virus signature of the virus information unit 304. If the instruction string matched a virus signature, the process continues at block 908. Otherwise, the process ends.

As shown in block 908, it is indicated that the instruction is associated with a virus. For example, the virus detection unit 212 indicates that the instruction is associated with a virus. In one embodiment, the virus detection unit 212 generates an interrupt indicating that the instruction is associated with a virus. In one embodiment, the associated interrupt handler flushes the processor pipeline and eliminates virus instructions from the processor and associated memory systems. From block 908, the process ends.

As noted above, Figures 10 and 11 describe concepts and operations for using a state machine to determine whether an instruction is associated with a virus. In particular, Figure 10 shows a state machine, while Figure 11 describes operations for moving between states of a state machine.

**Figure 10** is a block diagram of a state machine, according to exemplary embodiments of the invention. As shown in Figure 10, the state machine 1000 is a cyclic directed graph including four nodes, where each node is a state (e.g., START, S1, S2, and VIRUS represent states). The four states are START, S1, S2, and VIRUS. As shown in Figure 10, the four states are connected by a series of the directed edges (i.e., arrows), each directed edge having an associated condition. For example, the conditions for the three directed edges leading into the START state are “all other instructions.” The conditions for the directed edges leading out of the

START state are “instruction y” and “all other instructions.” In the state machine 1000, the “instruction y” condition represents a particular instruction. For example, “instruction y” could be an “XOR” instruction. Alternatively, “instruction y” could be an instruction with a specific pattern (e.g., XOR the contents of memory location 1000 with register EAX). The state machine 1000 begins at the START state and moves to other states when the various conditions are satisfied. Operations for moving through the states are described below in Figure 11.

**Figure 11** is a flow diagram illustrating operations for using a state machine in determining whether an instruction is associated with a virus, according to exemplary embodiments of the invention. The flow diagram 1100 will be described with reference to the exemplary state machine of Figure 10 and the exemplary processor of Figures 2-4. The flow diagram 1100 commences at block 1102, where an instruction is received. For example, the virus detection engine 302 receives an instruction from the level-one instruction cache 206. The process continues at block 1104.

At block 1104, progress is made to another state based on whether an instruction meets one of the state’s conditions. For example, referring to the state machine 1000 of Figure 10, the state machine proceeds from START to S1 when the instruction received at block 1102 matches “instruction y.” For all other instructions received at block 1102, the START state would cycle back to itself. As another example, the virus detection engine 302 proceeds from S1 to S2 when the received instruction matches “instruction x.” The process continues at block 1106.

At block 1106, the current state is stored. For example, the virus detection engine 302 stores the current state in the virus information unit 304. The process continues at block 1108.

As shown in block 1108, it is determined whether the current state is the VIRUS state. For example, referring to state machine 1000, the virus detection engine 302 determines whether the current state is the VIRUS state. If the current state is the virus state, the process continues at block 1110. Otherwise, the process continues at block 1102.

At block 1110, an indication that the instruction is associated with a virus is made. For example, the virus that unit 212 generates an interrupt for handling the virus. The process continues at block 1112.

As shown in block 1112, the current state is reset to the START state. For example, the virus detection engine 302 resets the current state to be the START state. From block 1112, the process ends.

As mentioned above, an interrupt handler can be provided to process and remove viruses. Figures 12 and 13 describe operations for providing such an interrupt handler, according to embodiments of the invention. In particular, Figure 12 describes operations for receiving an interrupt vector identifying an interrupt handler in a processor, while Figure 13 describes operations for transmitting the interrupt vector identifying an interrupt handler to the processor from an external source. According to embodiments of the invention, the operations for receiving and transmitting a interrupt vector identifying a virus processing interrupt handler may be performed using instructions of the processor's instruction set, which are specifically for communicating with the virus detection unit 212. For example, the virus information source could use special secure processor instructions for authenticating itself with the authentication unit 306.

**Figure 12** is a flow diagram illustrating operations for receiving an interrupt vector identifying a virus processing interrupt handler, according to exemplary embodiments of the invention. The flow diagram 1200 will be described with reference to the exemplary processors of Figures 2-4. The flow diagram 1200 begins at block 1202, where a request to register an interrupt vector identifying a virus processing interrupt handler is received. For example, hardware and/or software units (e.g., operating system software not shown in Figures 2-4), which are part of or executing on the processor 200, receive a request to register interrupt vectors identifying a virus processing interrupt handler. The process continues at block 1204.

At block 1204, a request for authentication data is transmitted. For example, the authentication unit 306 transmits a request for authentication data to the source

of the interrupt handler (e.g., a software process executing on the processor 200). The process continues at block 1206.

As shown in block 1206, authentication data is received and an attempt is made to authenticate the interrupt handler source. For example, the authentication unit 306 receives the authentication data from the interrupt handler source. The authentication unit 306 attempts to authenticate the interrupt handler source based on the authentication data. In one embodiment of the invention, the authentication unit 306 authenticates the interrupt handler source using a challenge-response authentication method. Alternative embodiments of the invention can use any suitable authentication method. The process continues at block 1208.

At block 1208, it is determined whether the authentication was successful. For example, the authentication unit 306 determines whether the interrupt handler source properly responded according to the implemented authentication technique (for example, challenge-response authentication). If the authentication was successful, control continues at block to 1210. Otherwise, the process ends.

At block 1210, the interrupt vector identifying the virus processing interrupt handler is received and installed. For example, hardware and/or software units (e.g., operating system software not shown in Figures 2-4), which are included in or executing on the processor 200, receive and install the virus processing interrupt handler to respond to interrupts generated by the virus detection unit 212. Additionally, the virus detection unit 212 receives an interrupt vector. In one embodiment, the interrupt vector corresponds to an interrupt descriptor table entry associated with the virus processing interrupt handler. In one embodiment, the virus detection unit 212 uses the interrupt vector to invoke the virus processing interrupt handler. For example, the virus detection unit 212 generates an interrupt that includes the interrupt vector. The interrupt causes the processor 200 to execute the virus processing interrupt handler because the virus processing interrupt is associated with the interrupt descriptor table entry indexed by the interrupt vector. In an alternative embodiment, the virus detection unit 212 receives different data for

enabling the virus detection unit 212 to invoke the virus processing interrupt handler (e.g., a memory address). From block 1210, the process ends.

**Figure 13** is a flow diagram illustrating operations for transmitting interrupt vector identifying a virus processing interrupt handler, according to exemplary embodiments of the invention. The flow diagram 1300 will be described with reference to the exemplary processor of Figures 2-4. The flow diagram 1300 commences at block 1302, where a request to install a interrupt vector identifying a virus processing interrupt handler is transmitted. For example, a virus processing interrupt handler source (e.g., a hardware or software unit external to the processor 200 containing a virus processing interrupt) transmits a request to install a interrupt vector identifying a virus processing interrupt handler. The process continues at block 1304.

At block 1304, a request for authentication data is received. For example, the virus processing interrupt handler source receives a request for authentication data from the authentication unit 306. The process continues at block 1306.

As shown in block 1306, authentication data is transmitted. For example, the virus interrupt handler source transmits authentication data to the authentication unit 306. The process continues at block 1308.

At block 1308, it is determined whether the authentication was successful. For example, the virus processing interrupt source receives an indication from the authentication unit 306 about whether the authentication was successful. If the authentication was successful, the process continues at block 1310. Otherwise, the process ends.

At block 1310, the interrupt vector identifying a virus processing interrupt handler is transmitted. For example, the virus processing interrupt handler source transmits the interrupt vector identifying a virus processing interrupt handler to hardware and/or software units (e.g., operating system software not shown in Figures 2-4) included in or executing on the processor 200 designated for receiving and installing interrupt handlers. From block 1310, the process ends.

Figures 14 and 15 describe operations for loading virus information including virus signatures and state information into a virus detection unit, according to embodiments of the invention. Figure 14 describes operations performed by the virus detection unit, while Figure 15 describes operations performed by a source of the virus detection information. According to  
5       embodiments of the invention, the operations for receiving and transmitting a virus information can be performed using instructions of the processor's instruction set, which are specifically for communicating with the virus detection unit 212. For example, the virus information source could use special secure processor  
10       instructions for authenticating itself with the authentication unit 306. Additionally, the virus information source could use special secure instructions for transmitting the virus information directly to the virus information unit 304.

**Figure 14** is a flow diagram illustrating operations for transmitting virus information to a virus detection unit, according to exemplary embodiments of the  
15       invention. The flow diagram 1400 will be described with reference to the exemplary processor of Figures 2-4. The flow diagram 1400 commences at block 1402, where a request to transmit virus information is received. For example, the virus information unit 304 receives a request to transmit virus information from a virus information source. In an alternative embodiment, the virus detection engine 302  
20       receives such a request. The process continues at block 1404.

As shown in block 1404, an attempt is made to authenticate the virus information source. For example, the virus information unit 304 instructs the authentication unit 306 to attempt to authenticate the virus information source. In one embodiment, the virus information source is a set of one or more hardware or  
25       software units external to the processor 200 containing virus information. In one embodiment, the virus information source is software executing on the processor 200, while in alternative embodiments, it is software executing on a separate processor, which is communicatively coupled to the processor 200. The process continues at block 1406.

At block 1406, it is determined whether the authentication was successful. For example, the authentication unit 306 determines whether the virus information source provided the proper authentication information (see discussion above for additional details about authentication). If the authentication was successful, the process continues at block 1408. Otherwise the process ends.

At block 1408, virus information including virus signatures and/or virus profiles is received and stored. For example, the virus information unit 304 receives virus information from a virus information source. In one embodiment of virus information includes virus signatures and/or virus profiles. In one embodiment, virus profiles include state information used by the virus detection engine 302 in determining whether instructions are associated with a virus (see discussion above). From block 1408, the process ends.

**Figure 15** is a flow diagram illustrating operations for transmitting virus information to a virus detection unit, according to exemplary embodiments of the invention. The flow diagram 1500 will be described with reference to the exemplary processors of Figures 2-4. The flow diagram 1500 commences at block 1502, where a request to transmit virus information is transmitted. For example, a virus information source transmits a request to transmit virus information to the virus detection unit 212. In one embodiment, virus information unit 304 receives this request. In alternative embodiment the virus detection engine 302 receives this request. The process continues at block 1504.

At block 1504, the request for authentication data is received. For example, the virus information source receives a request for authentication data from the authentication unit 306. In one embodiment, the request includes a challenge in accordance with a challenge-response authentication protocol. The process continues at block 1306.

As shown in block 1506, authentication data is transmitted. For example, the virus information source transmits authentication data to the authentication unit 306. In one embodiment, the virus information source transmits a response in

accordance with a challenge-response authentication protocol. The process continues at block 1508.

At block 1508, it is determined whether the authentication was successful. If the authentication was successful, the process continues at block 1510. Otherwise,  
5 the process ends.

As shown in block 1510, virus information including virus signatures and/or virus profiles is transmitted. For example, the virus information source transmits the virus information to the virus information unit 304. In alternative embodiment, the virus information source transmits the virus information to the virus detection  
10 engine 302. From block 1510, the process ends.

The virus detection system described above can be used to detect many types of viruses. In particular, the virus system can be used to detect polymorphic viruses. The following discussion of Figure 16 will describe the general behavior of polymorphic viruses.

15 **Figure 16** is a data flow diagram illustrating the general behavior of a polymorphic virus. In particular, the data flow diagram describes the behavior of a polymorphic virus in five stages. Initially, the polymorphic virus includes a virus body, mutation engine, and encryption/decryption loop. The virus body and mutation engine are initially encrypted, as indicated by the hash marks. During stage 1, the  
20 encryption-decryption loop decrypts the virus body and mutation engine, resulting in an entirely decrypted polymorphic virus (indicated by the removal of the hash marks).

During stage 2, the mutation engine "mutates" the encryption/decryption loop. That is, the mutation engine changes the encryption/decryption algorithm and  
25 code. During stage 3, the virus replicates the virus body and mutation engine (i.e., the virus creates new copies of the virus body and mutation). During stage 4, the mutated encryption-decryption loop encrypts the virus body and mutation engine. During stage 5, the virus attaches the newly created virus body (now encrypted), mutation engine (now encrypted), and mutated encryption/decryption loop to a host  
30 program. After stage 5, the host program is "infected" because it contains a copy of

the virus. After the host program is executed by a processor, the virus can further replicate itself, as described above.